REMARKS

The instant invention relates to a computer system providing an object-based virtual machine environment, in which middleware runs successive applications on a single virtual machine. The computer system noted includes storage for storing objects for running the aforementioned successive applications. The storage is logically divided into three heaps which are blocks of memory that belong to a program but have not yet been given a specific use. The heaps comprise:

a system heap which is not garbage collected;

a middleware heap which is garbage collected; and

a transient heap which is cleared inbetween successive applications.

Garbage collection refers to clearing out objects that are taking up space in memory but are no longer in use by the, e.g., JAVA program.

As noted above, in the present invention, the virtual machine has the system heap, the middleware heap and the private (or transient) heap. Classes are loaded and linked into the system heap, which will not be cleared or reset. Therefore class objects once loaded, linked and compiled into the system heap need not be reloaded, relinked or recompiled for subsequent applications. The middleware heap exists for middleware applications to store data that could be reset or must persist for subsequent applications. The private heap is used for application specific data and will always be cleared after an application runs and before the next application is scheduled into the virtual machine (hence it is generally referred to herein as the transient heap). This limitation has been included into the characterization of the "transient heap" defined in Claim 1.

In this approach, at the end of an application, the virtual machine stays up to execute the next application, rather than creating a virtual machine within a process to load and run an application, and then tearing down the entire process and runtime at the end of the application. Not creating the virtual machine environment per application increases the volume and throughput of applications a system can manage.

The present invention uses multiple heaps to retain persistent data and transient data. This use of multiple heaps enables a single virtual machine to be easily resettable, thereby avoiding the need to terminate and start a new virtual machine for each application. The use of multiple heaps also enables a single virtual machine to retain data and objects across multiple applications, thus avoiding the computing resource overhead of relinking, reloading, reverifying, and recompiling classes that have already been used by previous applications. In the preferred embodiment, the memory hierarchy also includes a system heap where classes are loaded, linked, verified, initialized and compiled. Subsequent applications can reuse the classes in the system heap and need not go through the overhead of reloading, linking, verifying and compiling them again. Middleware can create its persistent or resettable objects in the middleware heap. Any necessary garbage collection is preferably performed in between applications, thereby avoiding this overhead during the lifetime of an application, and so improving client response times. Application data that are used only during the lifetime of an application are created in the transient heap, which is cleared after every application.

A preferred embodiment of the virtual machine environment of the present invention uses multiple heaps to retain persistent data and transient data. This use of multiple heaps enables a single virtual machine to be easily resettable, thereby avoiding the need to terminate and start a new virtual machine for each application. The use of multiple heaps also enables a single virtual machine to retain data and objects across multiple applications, thus avoiding the computing resource overhead of relinking, reloading, reverifying, and recompiling classes that have already been used by previous applications. In the preferred embodiment, the memory hierarchy also includes a system heap where classes are loaded, linked, verified, initialized and compiled. Subsequent applications can reuse the classes in the system heap and need not go through the overhead of reloading, linking, verifying and compiling them again. Middleware can create its persistent or resettable objects in the middleware heap. Any necessary garbage collection is preferably performed in between applications, thereby avoiding this overhead during the lifetime of an application, and so improving client response times. Application data that are used only during the lifetime of an application are created in the transient heap, which is cleared after every application. In the preferred embodiment, a card table is used which is marked whenever a

possible reference to the transient heap is created. This enables a potentially faster route to allowing the virtual machine to reset the transient heap, compared to a full garbage collection, thereby increasing throughput of the virtual machine.

Thus in the preferred embodiment, a virtual machine has three types of heaps; a system heap, a middleware heap and a private heap. Classes are loaded and linked into the system heap, which will not be cleared or reset. Therefore class objects once loaded, linked and compiled into the system heap need not be reloaded, relinked or recompiled for subsequent applications. The middleware heap exists for middleware applications to store data that could be reset or must persist for subsequent applications. The private heap is used for application specific data and will always be cleared after an application runs and before the next application is scheduled into the virtual machine (hence it is generally referred to herein as the transient heap).

With this background in mind, Applicants respectfully submit that the references cited are not pertinent with respect to the present invention.

The Examiner is respectfully requested to reconsider the rejection of claims 1, 15 and 16 under 35 U.S.C. 102(a) as being anticipated by "Persistent Java Objects: A Proposal" hereinafter "Malhotra."

As set forth above in the explanation of the present invention, it is clear that the present invention relates to how applications can resuse heap. More specifically, the present invention teaches how multiple successive applications can resue the same heap.

From a reading of the Malhotra reference, one can only learn how to make a heap persistent. Unlike the present invention as presently claimed, Malhotra provides no teaching as to how multiple successive applications can reuse the same heap.

Applicants have amended Claims 1, 15 and 16 to more clearly define the invention and to distinguish the invention defined in these claims over the Malhotra disclosure.

As a result of the amendments to Claims 1, 15 and 16, the Malhotra reference does not provide a disclosure sufficient to anticipate the system used in accordance with Applicants' disclosure and claimed by them. In view of the lack of specificity of the Malhotra reference with respect to its teachings as applied to Applicants' invention as claimed, one can only conclude that the Malhotra reference does not now rise to the level required to qualify as an appropriate reference with respect to Applicants' invention; that is, wherein each and every element in Applicants' claims are anticipated.

> Further, the reference must describe the applicant's claimed invention sufficiently to have placed a person of ordinary skill in the field of the invention in possession of it. (Citations omitted) In re Lonnie T. Spada et al., 911 F.2d 705, 708 (Fed. Cir. 1990)

The lack of disclosure of the *middleware heap* alone in Malhotra is sufficient to negate this rejection.

The Examiner is respectfully requested to reconsider the rejection of Claims 1 - 7, 15 and 16 under 35 U.S.C. §103(a) as being unpatentable over in view of U.S. Patent 6,275,985 to Ungar in view of "*Concurrent Campacting Garbage Collection of a Persistent Heap* (hereinafter "O'Toole").

The Ungar system facilitates developing an application that implements garbage collection using a first compiler and proxy objects and then compiling the application with a second compiler that provides support for efficient garbage collection. During execution of the application, pointers within the system stack point indirectly to data objects through the proxy objects. The purpose of the Ungar system is to implement garbage collection in his application. Based upon a detailed

reading of the Ungar reference, Applicants respectfully emphasize that Ungar unequivocally teaches the use of a system heap and a transient heap for efficient means of garbage collection.

Yet, as a predicate for his rejection, the Examiner states on page 4 of the Official Action: "*...a system heap which is NOT garbage collected*" (Emphasis added), and cites Column 5, lines 1 - 45 in support of this assertion. The Examiner's assertion with respect to Ungar's system heap "not being a means of garbage collection" is clearly refuted at Column 5, line 30 etc. which states: "*System heap 312...contains proxy objects to help in the garbage collection process...*" Thus the Examiner's position in this respect is erroneous and, accordingly, the rejection is without foundation. The present invention and the Ungar reference are antithetical as far as their teachings are concerned as Applicants do not collect garbage in either the system heap or the transient heap and Ungar does collect garbage in the system heap and the transient heap.

O'Toole uses a heap to safely store transactional objects. It is important to note that O'Toole does not describe a middleware heap in his disclosure. O'Toole is in reality a means for replicating garbage collection for a persistent heap. The O'Toole disclosure teaches how a persistent heap is created. The "persistant heap" of O'Toole is analogous to the "system heap" defined by Applicants.

Essentially, Ungar and O'Toole teach new ways to garbage collect. Their emphasis is news ways to garbage collect.

Applicants invention differs from the combination of Ungar and O'Toole in that in Applicants' invention, there is a middleware heap which is garbage collected; further, there is a system heap and a transient heap, neither of which is garbage collected.

The Examiner is respectfully requested to reconsider the rejection of Claims 8 - 10 under 35 U.S.C. §103(a) as being unpatentable over in view of U.S. Patent 6,275,985 to Ungar and "Concurrent Campacting Garbage Collection of a Persistent Heap (hereinafter "O'Toole") in view of United States Patent 6,249,793 to Printezis, et al.

The arguments with respect to the Ungar and O'Toole references as set forth above are incorporated by reference at this site in response to the rejection including the Printezis, et al. reference.

There is no proper basis to combine the references Ungar, O'Toole and Printezis references as Ungar implements garbage collection in his system heap and O'Toole has no middleware heap. These features are contrary to Applicants' invention. Thus the defects present in these two inventions cannot rehabilitate these references when combined with Printezis, et al. to render the present invention obvious. The only element found in Printezis et al. which is relevant to the present invention is its disclosure of a card table-"*data structure*."

The Examiner is respectfully requested to reconsider the rejection of Claims 8 - 10 under 35 U.S.C. §103(a) as being unpatentable over in view of U.S. Patent 6,275,985 to Ungar and "Concurrent Campacting Garbage Collection of a Persistent Heap (hereinafter "O'Toole") in view of United States Patent 5,950,008 to van Hoff.

The arguments with respect to the Ungar and O'Toole references as set forth above are incorporated by reference at this site in response to the rejection including the van Hoff reference.

The van Hoff reference discloses a program interpreter for interpreting object oriented programs in a computer system having a memory that stores a plurality of objects and a plurality of methods. The van Hoff reference uses class loaders to describe what heap to create the objects in. Applicants' class loader as disclosed, is used to define where to place the storage objects--i.e., where the storage objects should be created.

In both the 35 U.S.C. 102(b) and 35 U.S.C. 103(a) rejections, the Examiner in the Official Action has selected concepts improperly from the references out of context as the basis for the rejections. In the rejections, especially with respect to 35 U.S.C. 103(a), the Examiner has improperly chosen excerpts as the basis for the rejection. The rejections are a piecemeal construction of the invention. Such piecemeal reconstruction of a prior art patent in light of the instant disclosure is contrary to the requirements of 35 U.S.C. 103 as set forth more specifically below.

Ungar and O'Toole alone, or in combination with Printezis or van Hoff, do not disclose or even imply the system and/or the method of the present invention. In the rejection, the Examiner is picking and choosing elements to the exclusion of what the references as a whole teach to one skilled in the art. To arrive at Applicants' invention, the person skilled in the art would have to randomly pick and choose among a substantial number of different elements and/or systems found in the references with absolutely no guidance whatsoever to direct him/her to the system and method claimed by Applicants. Based upon the skilled artisans knowledge of the properties of the systems disclosed in the prior references cited and their respective objectives and how they are implemented, it is unlikely that the Ungar reference would be used in combination with O'Toole and Printezis and van Hoff..

In order to analyze the propriety of the Examiner's rejections in this case, a review of the pertinent applicable law relating to 35 U.S.C. § 103 is warranted. The Examiner has applied the various references discussed above using selective combinations to render obvious the invention.

The Court of Appeals for the Federal Circuit has set guidelines governing such application of references. These guidelines are, as stated are found in Interconnect Planning Corp. v. Feil, 774 F.2d 1132, 1143, 227 USPQ, 543, 551:

> *When prior art references require selective combination by the court to render obvious a subsequent invention, there must be some reason for the combination other than hindsight gleaned from the invention itself.*

A representative case relying upon this rule of law is Uniroyal, Inc. v. Rudkin-Wiley Corp., 837 F.2d 1044, 5 USPQ 2d 1434 (Fed. Cir. 1988). The district court in Uniroyal found that a combination of various features from a plurality of prior art references suggested the claimed invention of the patent in suit. The Federal Circuit in its decision found that the district court did not show, however, that there was any teaching or suggestion in any of the references, or in the prior art as a whole, that would lead one with ordinary skill in the art to make the combination.

The Federal Circuit opined:

> *Something in the prior art as a whole must suggest the desirability, and thus the obviousness, of making the combination.* [837 F.2d at 1051, 5 USPQ 2d at 1438, citing Lindemann, 730 F.2d 1452, 221 USPQ 481, 488 (Fed. Cir. 1984).]

Applicants respectfully submit that there is no basis for the combination of the aforementioned references cited by the Examiner. Applicants have pointed out how the references teach in different directions. The Examiner has selected elements and steps from the cited references for the sake of showing the individual elements and/or steps claimed without regard to the total teaching of the references.

The Examiner in his application of the cited references is improperly picking and choosing. The rejection is a piecemeal construction of the invention. Such piecemeal reconstruction of the prior art patents in light of the instant disclosure is contrary to the requirements of 35 U.S.C. § 103.

> *The ever present question in cases within the ambit of 35 U.S.C. § 103 is whether the subject matter as a whole would have been obvious to one of ordinary skill in the art following the teachings of the prior art at the time the invention was made. It is impermissible within the framework of Section 103 to pick and choose from any one reference only so much of it as will support a given position, to the exclusion of other parts necessary to the full appreciation of what such reference fairly suggests to one of ordinary skill in the art. (Emphasis in original) In re Wesslau 147 U.S.P.Q. 391, 393 (CCPA 1965)*

This holding succinctly summarizes the Examiner's application of references in this case, because the Examiner did in fact pick and choose so much of each of the references cited to support his position and did not cover completely in the Office Action the full scope of what these varied disclosure references fairly suggest to one skilled in the art.

Further, the Federal Circuit has stated that the Patent Office bears the burden of establishing obviousness. It held this burden can only be satisfied by showing some objective teaching in the prior art or that knowledge generally available to one of ordinary skill in the art would lead that individual to combine the relevant teachings of the reference.

> *Obviousness is tested by "what the combined teachings of the references would have suggested to those of ordinary skill in the art."* In re Keller, 642 F.2d 413, 425, 208 USPQ 871, 881 (CCPA 1981). But it *"cannot be established by combining the teachings of the prior art to produce the claimed invention, absent some teaching or suggestion supporting the combination."* ACS Hosp. Sys., 732 F.2d at 1577, 221 USPQ at 933. [837 F.2d at 1075, 5 USPQ 2d at 1599.]

The court concluded its discussion of this issue by stating that teachings or references can be combined only if there is some suggestion or incentive to do so.

In the present case, the skilled artisan, viewing any or all of the references would be directed toward a totally different system than is called for in the present invention. The system emanating from the Ungar and O'Toole references would be a system that uses that has no middleware heap with NO garbage collection in the system heap and the transient heap. The references are teaching in opposite, or at least inconsistent directions. There is no proper basis to combine them.

Applicants have attempted in this response to amend the claims and to place these amended claims in a form which should result in their allowability. If the Examiner wishes to discuss via telephone the substance of any of the proposed claims contained herein with the intent of putting them into an allowable form, Applicants' attorney will be glad to speak with him at a mutually agreeable time and will cooperate in any way possible.

Any fees which result from the claims added herein should be charged to Deposit Account 50-0510.

In view of the arguments and modifications to the claims, allowance of this case is warranted. Such favorable action is respectfully solicited.

Respectfully submitted,

Thomas A. Beck
Reg. No. 20,816
26 Rockledge Lane
New Milford, CT 06776

I certify that this amendment is being deposited with the United States Postal Service on the date shown below with sufficient postage as first class mail in an envelope addressed to: *Assistant Commissioner of Patents, P.O. Box 1450. Alexandria, VA 22313-1450*

Signature_____ Date: August 16, 2004

Name: Thomas A. Beck

APPENDIX A
CLAIMS (STATUS)

1. (Currently amended) A computer system providing an object-based virtual machine environment, in which middleware runs successive applications on a single virtual machine, said system including storage for storing objects for running said applications, said storage being logically divided into three heaps:

a system heap wherein system classes for said single virtual machine are loaded into the system heap, thereby providing subsequent applications with the ability to use these classes without having to reload them and which system heap is not garbage collected;

a middleware heap which is garbage collected between successive applications; and wherein reusable objects other than class objects that must persist between successive applications, are stored in said middleware heap and

a transient heap which is cleared inbetween successive applications, and which has no garbage collected within the lifetime of an application said transient heap being used for storing applications objects that are used for only the duration of an application.

2. (Canceled) The computer system of claim 1, wherein system classes for the virtual machine are loaded into the system heap, thereby providing subsequent applications with the ability to use these classes without having to reload them

3. (Canceled) The computer system of claim 2, wherein reusable objects other than class objects that must persist between successive applications are stored in the middleware heap.

4. (Canceled) The computer system of claim 1, wherein the middleware heap is garbage collected between successive applications.

5. (Currently amended) The computer system of claim ~~4~~ 1, wherein only the portion of storage corresponding to the middleware heap is garbage collected between successive applications.

6. (Canceled) The computer system of claim 1, wherein the transient heap is used for storing applications objects that are used for only the duration of the application.

7. (Currently amended) The computer system of claim ~~6~~ 1, wherein at the end of an application, any objects in the transient heap that are eligible for use by the next application, and which are referenced by live objects in the system heap or middleware heap, are promoted to the middleware heap.

8. (Currently amended) The computer system of claim ~~6~~ 1, further comprising a card table, in which each card corresponds to a portion of the middleware heap, and said card is marked if the middleware heap potentially references an object in the transient heap.

9. (Previously presented) The computer system of claim 8, wherein each card corresponds to a memory region having a size greater than the minimum size for an object.

10. (Previously presented) The computer system of claim 9, wherein a card is marked whenever an object in the corresponding memory region is updated.

11. (Previously presented) The computer system of claim 1, further including a middleware classloader and an application class loader, wherein objects from classes loaded by the middleware classloader will be created in the middleware heap, and objects from classes loaded by the application classloader will be created in the transient heap.

12. (Previously presented) The computer system of claim 11, further including one or more system classloaders, wherein objects from classes loaded by the one or more system classloaders are created in the middleware heap or the transient heap depending on the current context.

13. (Previously presented) The computer system of claim 12, wherein the current context is middleware if the method being run derives from a class loaded by the middleware classloader, and application if the method being run derives from a class loaded by the application classloader.

14. (Previously presented) The computer system of claim 13, wherein if the method being run derives from a class loaded by said one or more system classloaders, then the current context retains the value it had immediately before the method was run.

15. (Currently amended) A method of operating a computer system providing an object-based virtual machine environment, in which middleware runs successive applications on a single virtual machine, said system including storage for storing objects for running said applications, said method comprising the steps of:

logically dividing the storage into three heaps:

a system heap wherein system classes for said single virtual machine are loaded into the system heap, thereby providing subsequent applications with the ability to use these classes without having to reload them,

a middleware heap,

and a transient heap;

performing garbage collection on ~~the~~ said middleware heap between successive applications; and wherein reusable objects other than class objects that must persist between successive applications, are stored in said middleware heap and performing garbage collection on said ~~the~~ transient heap, but not on ~~the~~ said system heap; and

clearing the transient heap inbetween successive applications, said transient heap being used for storing applications objects that are used for only the duration of an application and which has no garbage collected within the lifetime of an application.

16. (Currently amended) A computer program product comprising computer program instructions encoded on a computer readable media for loading into a computer system which provides an object-based virtual machine environment, in which middleware runs successive applications on a single virtual machine, said system including storage for storing objects for running said applications, said instructions causing the computer system to perform a method comprising the steps of:

logically dividing the storage into three heaps:

a system heap wherein system classes for said single virtual machine are loaded into the system heap, thereby providing subsequent applications with the ability to use these classes without having to reload them,

a middleware heap,

and a transient heap;

performing garbage collection on ~~the~~ said middleware heap between successive applications; and wherein reusable objects other than class objects that must persist between successive applications, are stored in said middleware heap and performing garbage collection on said ~~the~~ transient heap, but not on ~~the~~ said system heap; and

clearing the transient heap inbetween successive applications, said transient heap being used for storing applications objects that are used for only the duration of an application and which has no garbage collected within the lifetime of an application.